Capitolo 2

XML

2.1 Introduzione

XML (e**X**tensible **M**arkup **L**anguage) è un linguaggio di *markup* definito dal W3C (World Wide Web Consortium) nel 1998 utilizzato per descrivere dati in modo semplice e strutturato. Nasce come semplificazione dell'SGML (Standard Generalized Markup Language) un linguaggio standard ISO nel 1986.

La sintassi di XML assomiglia molto a quella di HTML (che è una applicazione del linguaggio SGML), però a differenza di quest'ultimo non ha *tag* predefiniti, ma è completamente estendibile. I tag sono gli elementi fondamentali dei linguaggi di markup con i quali i dati vengono strutturati. Ad esempio immaginiamo di voler inventare un formato di dati che ci permetta di gestire una semplice rubrica telefonica. I dati potrebbero essere memorizzati all'interno di un file testuale secondo la seguente formattazione:

```
Nome; Cognome; Telefono.
```

In questo modo ad esempio una rubrica di tre elementi apparirebbe come esegue:

```
Mario; Rossi; 031221222. Francesco; Neri; 123876453. Michele;
Bianchi; 022121222.
```

Questo modo semplice di descrivere i dati della nostra rubrica è in qualche modo un esempio di linguaggio di markup, utilizzando il carattere "**;**" come separatore fra il nome, il cognome e il numero di telefono, e il carattere "**.**" come separatore fra elementi distinti della rubrica. Lo scopo dei separatori è quello di permettere l'identificazione chiara ed univoca dei dati. Un file così

strutturato può essere facilmente importato da un opportuno programma in grado magari di aggiornarlo e fornire funzioni di ricerca per nome, numero di telefono, etc. XML fornisce strumenti per organizzare i dati in modo analogo al nostro esempio. Invece di utilizzare particolari caratteri per formattare i dati esso utilizza i *tag*. Ad esempio la nostra rubrica in XML potrebbe essere rappresentata in XML in questo modo:

```
<rubrica>
    <utente>
        <nome>Mario</nome>
        <cognome>Rossi</cognome>
        <telefono>031221222</telefono>
    </utente>
    <utente>
        <nome>Francesco</nome>
        <cognome>Neri</cognome>
        <telefono>123876453</telefono>
    </utente>
    <utente>
        <nome>Michele</nome>
        <cognome>Bianchi</cognome>
        <telefono>022121222</telefono>
     </utente>
</rubrica>
```

Anche se molto più prolisso del primo formato inventato, XML si presenta molto più leggibile; ma soprattutto appare subito chiaro il significato dei singoli elementi. Un tag, come vedremo, è in generale identificato da una coppia di parentesi angolari `<>`, all'interno della quale è presente un identificatore, come ad esempio `<nome>`.

HTML utilizza una sintassi simile a XML per strutturare il testo, ma a differenza di quest'ultimo, i suoi tag sono predefiniti (non se ne possono inventare di nuovi). Inoltre, il significato di ognuno dei suoi tag è completamente predefinito. La ragione è che HTML nasce come un linguaggio di presentazione dei dati sul video. Ad esempio in HTML ci sono tag che permettono di fare apparire il testo in grassetto, corsivo, etc. In generale i tag forniscono informazioni che i browser utilizzano per presentare il testo sullo schermo. I tag di HTML quindi non danno nessuna informazione su cosa rappresentano i dati al loro interno, ad esempio un elemento di una bibliografia in HTML potrebbe apparire come segue:

```
<h1> Bibliografia </h1> <p> <i> Appunti del corso di Biblioteche
Digitali</i>, C. Gennaro, P. Savino <br> Università di Pisa, 2004
```

Il fatto che il titolo dell'elemento bibliografico sia scritto in corsivo (utilizzando il tag `<i>`) non permette di capire che si tratta appunto di un titolo. In XML l'esempio di sopra potrebbe essere rappresentato nel seguente modo:

```
<bibliografia>
    <dispensa>
        <titolo>Appunti del corso di Biblioteche Digitali</titolo>
        <autore>Claudio Gennaro</autore>
        <autore>Pasquale Savino</autore>
        <editrice>Università di Pisa</editrice>
        <anno>2004</anno>
    </dispensa>
</bibliografia>
```

Si noti la maggior espressività di XML dovuta all'introduzione del tag `<bibliografia>` che permette di identificarne esattamente il suo contenuto. Nel caso di HTML il termine *Bibliografia* appariva semplicemente come titolo della sezione.

È importante sottolineare che il problema della semantica attribuita ai tag non viene risolto dalla semplice sintassi XML. Il fatto che un certo tag si chiami `<titolo>`, riesce a suggerire ad un lettore "umano" che probabilmente abbiamo a che fare con il titolo di qualcosa, ma questo non è sufficiente a definirne una semantica in modo univoco. Soprattutto non dice nulla ad un generico programma che eventualmente deve elaborare il documento. XML rappresenta solo un mezzo per la creazione di linguaggi standard per strutturare dati, ma la semantica dei tag è definita esternamente al linguaggio ed in qualche modo inglobata nei programmi che utilizzano i dati XML.

Inoltre, XML non dà nessuna indicazione su come i dati devono essere utilizzati, rappresentati o elaborati. Come esempio si consideri il caso in cui abbiamo definito un linguaggio per descrivere spartiti musicali in XML. Ci saranno tag che definiscono il tempo della partitura, la tonalità delle note, la loro durata etc. Quindi la semantica dei tag di un tale linguaggio basato su XML, è ben chiara, stiamo parlando di note musicali. Ma cosa possiamo farci con un file XML di questo tipo? Ad esempio possiamo farlo suonare al computer, oppure possiamo rappresentarlo sul video come uno spartito musicale per musicisti, oppure ancora trasformarlo in un file midi per essere eseguito poi da un tastiera elettronica. L'uso che si fa di un documento XML è a discrezione, per così dire, dell'applicazione che lo impiega. Invece

nel caso di HTML, un documento può essere solo rappresentato sul video o su carta.

Esistono oggi molti linguaggi basati XML, alcuni di essi hanno uno scopo semplice, strutturare dati tipicamente per il WEB, come ad esempio `MathML`, che permette di definire formule matematiche, altri sono più difficili da comprendere ad esempio XML-Schema. Daremo di quest'ultimo una breve descrizione alla fine del capitolo.

Anche se XML è nato per il WEB, il suo impiego è diventato ampio. Di per sè, XML non rappresenta niente di particolarmente innovativo, ma il fatto che tutte le applicazioni oggi, videoscrittura, basi di dati, etc. lo utilizzano significa che ormai è stato pienamente accettato come standard da tutti. Non ultimo Microsoft ha investito moltissimo su XML, partecipando alla definizione dello standard, ed vari linguaggio basati su XML. Per quanto riguarda il WEB, anche se le pagine potrebbero essere scritte in XML, oggi si preferisce utilizzare HTML (o la sua versione in XML, XHTML) per scrivere le pagine WEB, visto che è supportato da tutti browser. XML è quindi in qualche modo complementare ad HTML e il suo scopo non è quello di sostituirlo.

2.2 Sintassi di XML

In questa sezione daremo una breve descrizione della sintassi XML senza peraltro la pretesa di essere esaustivi.

Un documento XML si dice ben formato (well-formed) quando è sintatticamente corretto. Alcuni esempi di errori di sintassi come vedremo possono essere tag aperti ma non chiusi.

Il prologo

Un documento XML è costituito da un prologo e da un elemento radice che contiene i dati XML. Il prologo non contiene dati veri e propri, ma contiene informazioni sul documento XML (per esempio la versione XML, sul set di caratteri utilizzato, sulla struttura, etc.), quindi metadati, che servono ai programmi che operano su di esso per utilizzarlo correttamente. In particolare il prologo si trova sempre all'inizio del documento e può contenere, una dichiarazione XML, commenti, istruzioni di processo e i *Document Type Definitions* (DTD), separati eventualmente da spazi bianchi.

Tutti i documenti XML dovrebbero cominciare con una *dichiarazione XML*. Essa è racchiusa fra le stringhe `<?xml` e `?>` e tipicamente nei documenti si presenta come segue

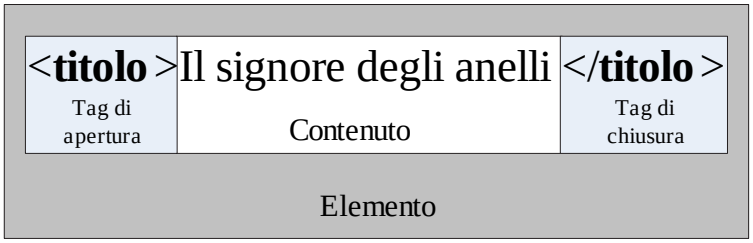

Figura 2.1: Struttura di un elemento XML.

`<?xml version=1.0 encoding=UTF-8?>`

`version` indica la versione di XML (1.0) ed `encoding` il set di caratteri utilizzato, in questo caso UTF-8 (il caso più comune).

I commenti in XML sono racchiusi fra le stringhe `<!--` e `-->`, ad esempio: `<!-- questo è un commento -->`.

I commenti possono essere messi ovunque nel documenti a patto di non comparire prima della dichiarazione XML, di non spezzare gli elementi di markup (come i tag), e di non contenere la stringa `--`. I commenti sono sempre ignorati dai programmi che leggono i documenti XML.

Le istruzioni di processo sono racchiuse fra le stringhe `<?` e `?>`, così come la dichiarazione XML. Esse sono utilizzate per comunicare informazioni a programmi esterni. La loro forma standard di un'istruzione di processo è: `<?`*`target ...istruzione...`* `?>`. Il *target* è obbligatorio e indica un'applicazione. La parte *...istruzione...* comunica cosa deve fare il programma *target*. Le istruzioni di processo possono in realtà trovarsi ovunque all'interno del documento XML (non solo nel prologo). Ecco un esempio di istruzione di processo legale: `<?perl lower-to-upper-case ?>`.

Dei DTD si parlerà più avanti in questo capitolo.

Gli elementi

I dati XML veri e propri seguono il prologo e sono organizzati in *elementi*. Un elemento è costituito da un tag aperto e chiuso, come ad esempio `<titolo>Il signore degli anelli</titolo>`. Un elemento è quindi formato da un tag di apertura del tipo `<nomedeltag>` e da un tag di chiusura del tipo `</nomedeltag>` (vedi Figura 2.1).

Inserire un tag di apertura senza il corrispondente tag di chiusura è un errore sintattico (diversamente da quello che accade per HTML che accetta tag del tipo `<p>` senza il corrispondente `</p>`). Il nome del tag può essere

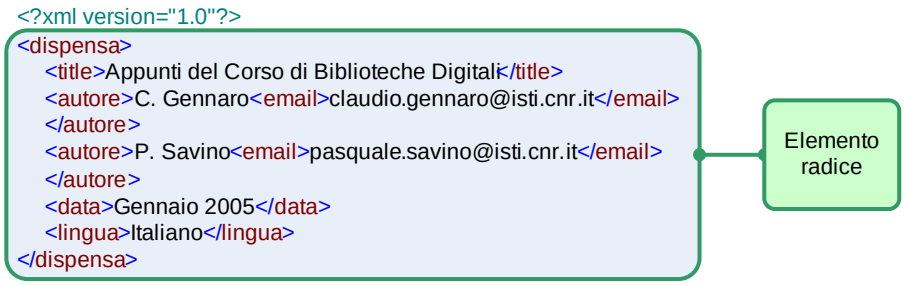

Figura 2.2: Semplice esempio di un documento XML.

costituito da stringe di lettere e numeri, ma non può contenere spazi, non può iniziare per XML e non può iniziare con un numero o con un carattere di punteggiatura. Inoltre, dato che XML distingue i caratteri maiuscoli da quelli minuscoli, un elemento `<titolo>` non è equivalente a `<Titolo>`.

Gli elementi possono contenere solo testo (come nel caso di Figura 2.1) oppure contenere altri elementi (che chiameremo sotto–elementi), ad esempio:

```
<persona>
    <nome>Claudio</nome>
    <cognome>Gennaro</cognome>
</persona>
```

oppure ancora contenere un misto di testo ed elementi:

```
<titolo>Il <i>signore</i> degli anelli</titolo>
```

Un caso speciale è l'elemento vuoto, vale a dire che non contiene nulla, può essere abbreviato. Ad esempio `<p></p>`, in XML è equivalente a `<p/>` (da non confondere con il tag di chiusura `</p>`).

È importante sapere che un documento XML corretto contiene un solo elemento principale (chiamato *root element* cioè elemento radice).

In Figura 2.2 è mostrato un piccolo esempio di documento XML completo. L'esempio mostra come rappresentare i metadati di una dispensa. Gli elementi di tipo `autore` sono di tipo misto e contengono all'interno una parte di testo ed un elemento di tipo `email`.

Gli attributi

Gli attributi forniscono informazioni addizionali sugli elementi XML. Si potrebbe dire che gli elementi sono i nomi di XML, mentre gli attributi rappresentano gli aggettivi. Ad esempio l'email dell'autore dell'elemento `dispensa` potrebbe essere definito usando un attributo:

```
<autore email="claudio.gennaro@isti.cnr.it"> Claudio
Gennaro</autore>
```

Un attributo si trova all'interno del tag di apertura di un elemento ed è espresso nella forma *nome_attributo=valore*. Ad un elemento possono essere associati uno o piu attributi separati da uno spazio.

La scelta di inserire un dato in un documento come contenuto di un elemento piuttosto che come contenuto di un attributo, dipende esclusivamente dal creatore del documento, vale a dire che non esiste una regola generale. Ad esempio il seguente elemento che definisce una persona:

```
<persona sesso="maschio">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
</persona>
```

può benissimo essere implementato anche senza attributi:

```
<persona>
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
    <sesso>maschio</sesso>
</persona>
```

L'unica attenzione è quella di codificare dati omologhi nello stesso modo, ad esempio questa versione appare "stonata" a chiunque:

```
<persona cognome="rossi">
    <nome>Mario</nome>
    <sesso>maschio</sesso>
</persona>
```

Il valore di un attributo è racchiuso tipicamente tra una coppia di doppi apici (”). Lo standard XML accetta però anche gli apici singoli (apostrofi) (’) al posto dei doppi apici. In generale le stringhe degli attributi XML sono chiamate *dati letterali* (*string literals* in inglese). Esiste una restrizione per

cui il carattere utilizzato come delimitatore non può essere contenuto nel dato letterale, ed in particolare se il singolo apice appare nella stringa il doppio apice deve essere usato come delimitatore del letterale, e viceversa. Ecco alcuni esempi di dati letterali corretti:

```
"esempio" 'esempio' "esempio 'corretto'"
```

I seguenti dati letterali sono invece scorretti:

```
"esempio' 'questo esempio e' "scorretto"'
```

Più in generale, in un dato letterale o nel contenuto di un elemento è possibile usare quasi tutti i caratteri. Sono esclusi: &, < (minore), > (maggiore), " (doppio apice) e ' (apostrofo), poiché riservati al linguaggio XML e non utilizzabili quindi nei dati letterali. Per utilizzarli nel contenuto degli attributi o degli elementi XML bisogna utilizzare i *riferimenti alle entità*, che iniziano con il carattere & e terminano con un punto e virgola. In generale le entità permettono di definire alcune parti che poi i programmi che utilizzano XML sostituiranno. Alcune entità sono predefinite nel linguaggio XML e permettono di inserire quei caratteri che altrimenti sarebbero inutilizzabili. Le entità predefinite sono le seguenti:

entità	significato
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Tipicamente i programmi che visualizzano documenti XML come i browser sostituiscono automaticamente le entità nei loro corrispondenti caratteri.

XML permette di definire nuove entità. Si veda la sezione sui DTD per un approfondimento sulle entità.

Spesso accade che alcuni nomi di elementi o attributi usati all'interno di un documento XML entrino in conflitto. Ad esempio il nome dell'elemento `titolo`, potrebbe indicare il titolo di un libro o di un dipinto. XML fornisce un utile meccanismo in grado di definire degli spazi di nomi (chiamati *namespace*) per risolvere queste ambiguità. Un namespace consiste in una collezione di nomi che possono essere usati come elementi e nomi di attributi. I nomi del namespace vengono identificati utilizzando la seguente sintassi: `ns-prefix:local-name`. Ad esempio nel nostro caso potremmo distinguere i nostri tag come `<libro:titolo>` e `<dipinto:titolo>`. Un namespace deve essere dichiarato attraverso l'attributo `xmlns` prima di poterlo utilizzare

all'interno di elemento. Ad esempio possiamo definire il namespace *libro* nel seguente modo:

```
<biblioteca xmlns:libro="http://www.esempio.org/1999/libro">
    <libro:titolo>...</libro:titolo>
</biblioteca>
```

L'attributo `xmlns` definisce il namespace libro identificandolo univocamente con un URL, che nel nostro esempio è `http://www.esempio.org/1999/libro`, ma può anche essere un URI (Uniform Resource Identifiers)[1] (Unified Resource Identifier) qualsiasi. A volte si utilizza l'URL di un documento DTD come URI.

2.3 I DTD

Quando un documento XML deve essere utilizzato da uno specifico software il programmatore analizza il documento con un programma particolare chiamato *Parser*, che ha lo scopo di verificarne la correttezza. Come è stato precisato sopra, un documento corretto, ossia ben formato, non ha errori di sintassi. Il che significa che i tag sono tutti chiusi, innestati correttamente (il tag di chiusura di un elemento più interno si trova prima del tag di chiusura dell'elemento che lo contiene). Inoltre esiste un solo elemento radice, e i nomi di elementi e attributi sono conformi allo standard XML. Il parser esegue tutti questi controlli e se qualcosa va storto restituisce solitamente un codice di errore che permette di individuare l'errore sintattico. Eseguire il *parsing* dei documenti XML è importante in quanto l'elaborazione che ne segue potrebbe avere dei risultati inpredicibili se il documento non è ben formato.

Oltre alla verifica della correttezza spesso è necessario riconoscere se un documento XML è conforme ad una struttura predefinita. Ad esempio supponete di voler fare in modo che i vostri documenti XML siano del tipo persona visto precedentemente:

```
<persona sesso="maschio">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
</persona>
```

[1] URI è la generalizzazione di un URL. La differenza è che un URI può identificare oggetti anche non correlati ai protocolli Internet esistenti. Si pensi al caso di about:netscape. In pratica un URI è un URL più generale che non fa riferimento a un preciso schema di denominazione.

Volete quindi evitare che compaiano elementi diversi da quelli definiti, essere sicuri che gli elementi siano nell'ordine prestabilito, etc. Ad esempio il seguente documento

```
<persona sesso="maschio">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
    <datadinascita>15-05-1956</datadinascita>
</persona>
```

è corretto ma potrebbe essere non valido, se l'elemento `<datadinascita>`non è stato definito nel suo DTD. Si parla in questo caso di *validazione* di un documento XML (rispetto ad una struttura predefinita). La validazione viene di solito eseguita da speciali parser chiamati appunto validatori. Ma come si fa a definire la struttura del documento? XML 1.0 fornisce un linguaggio apposito (tecnicamente è una grammatica formale) chiamato DTD (Document Type Definition) che permette di definire un vocabolario di nomi (del tipo `nome`, `cognome`, etc.), ed alcune regole obbligate di composizione tra gli elementi (del tipo "questo elemento viene sempre prima di quest'altro"). L'operazione di validazione permette di confrontare un documento XML con le sue regole e di identificare e segnalare eventuali difformità.

I vantaggi di utilizzare il DTD sono molteplici:

- usando i DTD ogni documento contiene una descrizione del proprio formato;
- gruppi di utenti possono accordarsi sull'uso di DTD comuni per facilitare lo scambio dei documenti;
- le applicazioni possono usare un DTD standard per verificare che i dati ricevuti dall'esterno sono validi;
- si possono utilizzare i DTD per verificare che i dati prodotti abbiano la struttura corretta;
- un'applicazione per la produzione di documenti XML (editor) può sfruttare il DTD per creare un'interfaccia di inserimento dati conforme con il modello DTD, così da evitare errori di inserimento all'utente;

Un esempio di DTD

Prima di affrontare la trattazione del DTD, vediamo un semplice esempio che ne mostri le potenzialità. Vogliamo definire un DTD per validare un

elemento di tipo `<persona>` come nell'esempio sopra, ecco come potrebbe essere realizzato:

```
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT persona (nome, cognome)>
<!ATTLIST persona
          sesso CDATA #REQUIRED>
```

Le prime due righe del DTD dichiarano i due elementi `cognome` e `nome`, come elementi che contengono testo (`#PCDATA`). La terza riga invece definisce `persona` come un elemento che contiene i due elementi precedentemente definiti. Le ultime due righe del DTD dichiarano che l'elemento `persona` contiene un attributo testuale (`CDATA`). L'oggetto `#REQUIRED` indica che l'attributo è obbligatorio.

Il Document Type Declaration

Il DTD così definito può essere associato al documento XML in due modi diversi, esternamente al documento o internamente, attraverso una dichiarazione *Document Type Declaration* (da non confondere con Document Type Definition DTD). Nel caso di dichiarazione esterna il DTD (come quello dell'esempio sopra) viene inserito in un file di estensione `dtd`. Ad esempio supponiamo che il nostro DTD è memorizzato nel file `persona.dtd`. All'interno del prologo del documento XML corrispondente comparirà una dichiarazione `DOCTYPE`, come segue:

```
<?xml version="1.0"?>
<!DOCTYPE persona SYSTEM "persona.dtd">
<persona sesso="maschio">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
</persona>
```

In generale la sintassi di un Document Type Declaration è la seguente:
`<!DOCTYPE` *`nome-elemento-radice`* `SYSTEM` *`sorgente_del_dtd`*`».` *nome-elemento-radice* indica il nome dell'elemento radice descritto nel DTD, e *sorgente_del_dtd* specifica il nome del file (URL) che contiene il DTD. La parola chiave `SYSTEM` significa che il DTD è esplicitamente individuabile dal parser. Se invece è di tipo `PUBLIC` allora la sorgente è un generico URI. In questo caso si tratta di DTD "ben noti" che il parser sa (o dovrebbe sapere) come recuperare.

Se si vuole inserire il DTD internamente al documento XML si utilizza sempre il Document Type Declaration, inserendo il DTD dentro la dichiarazione come qui di seguito:

```
<?xml version="1.0"?> <!DOCTYPE persona [ <!ELEMENT cognome
(#PCDATA)> <!ELEMENT nome (#PCDATA)> <!ELEMENT persona (nome,
cognome)> <!ATTLIST persona
          sesso CDATA #REQUIRED>
]> <persona sesso="maschio">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
</persona>
```

Si noti come il tutto il DTD è contenuto tra le parentesi quadrate `[ ]`.

Il linguaggio dei DTD è molto ricco e flessibile. In questo capitolo daremo solo uno sguardo alle funzionalità principali del linguaggio.

Elementi

Gli elementi nel DTD vengono definiti, come è stato visto, utilizzando una dichiarazione `<!ELEMENT ...>`, la cui sintassi è la seguente:
`<!ELEMENT` *name content-model*`>`.
Dove *name* è il nome dell'elemento (ad es. titolo, persona, ecc.), e *content-model* permette di specificare quale tipo di contenuto può essere incluso nell'elemento, quanti possono essere i suoi elementi e in che ordine vanno inseriti. I casi più semplici di content-model sono `EMPTY` e `ANY`. Il primo specifica che l'elemento deve essere vuoto (ad esempio `<x/>`) e il secondo qualsiasi cosa (ossia elementi, testo) purché corretta sintatticamente.

Il content-model di un elemento più generale è composto da un insieme di parentesi che racchiudono combinazioni di nomi, operatori e la parola chiave `#PCDATA`. Gli operatori utilizzabili sono descritti nella seguente tabella:

Operatore	significato
,	(virgola) sequenza rigida
|	(pipe) scelta

L'operatore "`,`" permette di specificare una sequenza di elementi figli nell'ordine in cui appaiono, come nell'esempio già visto:
`<!ELEMENT persona (nome, cognome)>`. L'operatore "|" significa "o l'uno o l'altro", ad esempio:
`<!ELEMENT ditta (codice_fiscale | partita_iva)>`.
In questo caso l'elemento `ditta` pùo contenere l'elemento figlio `codice_fiscale`

o in alternativa `partita_iva`, tutti e due saranno ritenuti validi dal parser. Utilizzando le parentesi è possibile combinare a piacimento i due operatori in modo da formare regole più complesse. Ad esempio:
`<!ELEMENT persona ((nome, cognome) | (cognome, nome))>`,
significa che gli elementi figli `nome` e `cognome` possono trovarsi in qualsiasi ordine. DTD permette di specificare anche il numero di elementi figli attraverso i seguenti operatori di cardinalità

Operatore	significato
?	opzionale
*	zero o più
+	uno o più

L'operatore "?" posto dopo il nome di un elemento indica che è opzionale, ad esempio:
`<!ELEMENT contatto (telefono, cellulare?, fax?)>`
Gli elementi figli `cellulare` e `fax` possono anche non comparire. L'operatore "*" indica che l'elemento può comparire da zero ad un numero arbitrario di volte, ed infine "+" indica che l'elemento deve comparire almeno una volta. Vediamo un esempio più articolato di uso degli operatori di cardinalità:
`<!ELEMENT equipaggio (pilota+, (hostess | steward)*)>`
In questo caso l'elemento equipaggio può essere formato da almeno un elemento `pilota` e da un certo numero di `hostess` e `steward`, come nel seguente esempio:

```
<equipaggio>
    <pilota>...</pilota>
    <pilota>...</pilota>
    <pilota>...</pilota>
    <hostess>...</hostess>
    <steward>...</steward>
    <steward>...</steward>
</equipaggio>
```

Supponiamo invece di voler definire un elemento con contenuto di tipo misto, come ad esempio nel caso di HTML in cui il testo può essere in corsivo, usando il tag `<i>` o in grassetto, usando il tag `<b>`, potremmo scrivere:
`<!ELEMENT testo (#PCDATA | i | b)*>`
Un esempio valido dell'elemento testo così definito potrebbe essere il seguente:

```
<testo> Normale <i>corsivo</i> normale <b>grassetto</b> normale.
</testo>
```

In accordo alle specifiche di XML 1.0 la parola chiave `#PCDATA` deve comparire sempre per prima usando l'operatore `|` quando si definisce contenuto di tipo misto.

Attributi

Gli attributi permettono di completare gli elementi associando ad essi alcuni semplici dati aggiuntivi. Con DTD possono essere definiti usando una dichiarazione del tipo `<!ATTLIST ...>`, la struttura è la seguente:

```
<!ATTLIST nome_elemento
nome_attributo1 tipo_attributo1 default1
nome_attributo2 tipo_attributo2 default2
.
.
.
>
```

Ogni dichiarazione `<!ATTLIST ...>` permette di definire uno o più attributi associati allo stesso elemento (indicato sopra come *nome_elemento*). Ogni attributo è definito da tre elementi: il nome (*nome_attributo*), il tipo di dato (*tipo_attributo*) e un valore di default (*default*). Il tipo di dato può assumere i valori descritti in Tabella 2.1. I possibili valori di defaults sono descritti in tabella 2.2.

Parliamo prima di quest'ultimi. Quando un attributo è dichiarato come `#REQUIRED` allora deve apparire obbligatoriamente nell'elemento in cui è specificato. Se invece è dichiarato `#IMPLIED` è opzionale. Attraverso la parola chiave `#FIXED` può essere specificato un valore fisso che deve assumere l'attributo, che, nel caso di omissione nel documento XML, viene assunto automaticamente dal parser.

Nel caso in cui venga omessa la parola chiave `#FIXED`, ma venga indicato solo un valore di Default (ultimo caso in tabella), allora l'attributo potrà assumere anche valori diversi da quello specificato, se invece non compare nel documento il parser utilizzerà quello indicato. Ad esempio:

```
<!ATTLIST Sfondo
          colore CDATA "bianco" #FIXED
          immagine CDATA "tramonto.gif"
          >
```

Se nell'elemento `Sfondo` non viene specificato l'attributo `colore` allora il parser utilizzerà *`bianco`* come valore predefinito. In tutti i casi, valori diversi

Tipo di attributo	Utilizzo
`CDATA`	L'attributo può contenere solo dati di tipo testo
`ENTITY`	L'attributo deve fare riferimento a un'entità binaria esterna dichiarata nel DTD
`ENTITIES`	È equivalente all'attributo ENTITY, ma consente l'utilizzo di più valori separati da spazi
`ID`	Il valore dell'attributo deve essere un identificatore unico all'interno dello stesso documento
`IDREF`	Il valore deve essere un riferimento a un ID dichiarato in un altro punto del documento. Se l'attributo non corrisponde al valore dell'ID specificato, il parser produrrà un errore
`IDREFS`	È equivalente all'attributo IDREF, ma consente l'utilizzo di più valori separati da spazi.
`NMTOKEN`	L'attributo consiste in una qualsiasi combinazione di caratteri del token (una stringa senza spazi) del nome, rappresentati da lettere, numeri, punti, trattini, due punti o caratteri di sottolineatura
`NMTOKENS`	È equivalente all'attributo NMTOKEN, ma consente l'utilizzo di più valori separati da spazi
`NOTATION`	Il valore dell'attributo deve fare un riferimento a un'annotazione dichiarata in un altro punto del DTD. La dichiarazione può anche essere costituita da un elenco di annotazioni. Il valore deve corrispondere a una delle annotazioni dell'elenco. Ogni annotazione deve avere la relativa dichiarazione nella DTD
`Enumerated`	Il valore dell'attributo deve corrispondere a uno dei valori inclusi

Tabella 2.1: Tipi di dati degli attributi

Default attributo	Significato
`#REQUIRED`	L'attributo deve comparire in ogni istanza dell'elemento
`#IMPLIED`	Questo attributo è opzionale
`#FIXED` *`valorefisso`*	Questo attributo deve avere il valore *valorefisso*. Se l'attributo non è incluso nell'elemento, viene assunto il valore *valorefisso*
`solo un valore di Default`	Identifica un valore predefinito per un attributo. Se l'elemento non include l'attributo, viene stabilito il valore default Se l'attributo compare può contenere un altro valore.

Tabella 2.2: Valori di Default degli attributi

producono un errore da parte del parser. Per quanto riguarda l'attributo `immagine` se viene omesso viene assunto come valore *`tramonto.gif`*.

XML permette di definire svariati tipi di dati (vedi Tabella 2.1). Il tipo `CDATA` indica un valore dell'attributo di tipo testo, ad esempio:
`<elemento testo="questo è valido">...</elemento>`.

I tipo `ID`, `IDREF` e `IDREFS`, sono utili per specificare identificatori. Quando un attributo è di tipo `ID`, il valore da esso assunto deve essere unico all'interno dello stesso documento XML. Questo è utile quando si vuole essere sicuri dell'unicità del dato inserito nell'attributo, come nel seguente esempio:

```
...

<!ATTLIST libro
    collocazione ID #REQUIRED
    >
...

<libro collocazione="SF234">
    <titolo>...</titolo>
    <autore>...</autore>
    ...
</libro>
```

Il parser verificherà per noi che non ci siano due elementi `libro` con lo stesso valore in `collocazione`. Bisogna precisare che un elemento può avere al più un solo attributo di tipo `ID`. Inoltre gli attributi `ID` sono sempre `#REQUIRED` o `#IMPLIED`, ma mai `#FIXED` o Defaults. Infatti non avrebbe senso avere degli indentificatori unici che abbiano valori predefiniti. Gli `IDREF` sono utili per

indirizzare attributi di tipo ID definiti da qualche altra parte nel documento. Per capirne l'utilità si supponga di avere la seguente definizione di elemento del DTD dell'esempio precedente:

```
<!ELEMENT Prestito (nome, cognome)> <!ATTLIST Prestito
    collocazione IDREF #REQUIRED
    >
```

In tal caso l'elemento `Prestito` può fare riferimento alla collocazione del libro senza dover ripetere tutto l'elemento libro.

```
...

<libro collocazione="SF234">
    <titolo>...</titolo>
    <autore>...</autore>
    ...
</libro>

<Prestito collocazione="SF234">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
</Prestito>
```

Per capire gli attributi di tipo ENTITY dobbiamo prima parlare delle entità. Nella sezione precedente abbiamo parlato delle entità predefinite, come ad esempio `&` che viene sostituito automaticamente nel carattere `&` dal parser. XML permette di definire nuove entità utilizzando una dichiarazione di tipo `<!ENTITY ...>`. Le entità hanno tutte un contenuto e sono identificate da un nome. Le entità si suddividono in due categorie: interne ed esterne. Le entità interne, costituite sempre da valori di testo, vengono sempre definite in un DTD e forniscono riferimenti di collegamento al testo che deve essere sostituito in un documento. Le entità esterne, invece, possono essere costituite da testo o da qualsiasi altro tipo di dati (ad esempio dell'origine di un'immagine) vengono sempre memorizzate in un file esterno, e forniscono riferimenti di collegamento ai file di dati che devono essere sostituiti in un documento. In realtà la strutturazione delle entità è più complessa, si veda un testo specifico su XML per maggiori dettagli.

Ad esempio definendo nel DTD la seguente entità interna:

```
<!ENTITY Autore "Claudio Gennaro">
```

possiamo inserire nel documento XML corrispondente un riferimento all'entità come di seguito:

```
<Testo> Questo testo è stato scritto da &Autore;</Testo>
```

Il parser provvederà alla sostituzione dell'entità con il valore specificato al momento dell'analisi del documento.

Nelle entità esterne il testo da sostituire si può trovare in un file esterno. In questo caso la dichiarazione ha la seguente forma:

```
<!ENTITY libro SYSTEM "http://www.cnr.it/pub/libro.html">
```

La parola chiave `SYSTEM` è usata per indicare una sorgente esterna identificata da un URL.

Tornando agli attributi, quelli definiti come `ENTITY`, possono far riferimento ad entità esterne, come esemplificato qui di seguito:

```
<!ATTLIST Libro copertina ENTITY #REQUIRED> <!ENTITY figura1
SYSTEM "http://www.cnr.it/imgs/book1.gif" NDATA GIF>

<Libro copertina="figura1">
    ...
</Libro>
```

In questo caso l'immagine della copertina del libro, specificata con l'attributo `copertina`, è esterna al file. La parola chiave `NDATA` indica una cosiddetta annotazione, ossia la specifica dell'applicazione da usarsi per elaborare l'entità non analizzabile. In poche parole la precedente dichiarazione deve essere accompagnata dall'opportuna annotazione che indica come elaborare la classe di entità GIF, ad esempio:

```
<!NOTATION GIF SYSTEM "gifview.exe">
```

Gli attributi di tipo `NMTOKEN` possono contenere una stringa di una parola sola (quindi senza spazi) contenenti lettere, numeri, punti, trattini, due punti o caratteri di sottolineatura. Ad esempio un numero di telefono:

```
...
<!ATTLIST Ufficio telefono NMTOKEN>
...
<Ufficio telefono = "0039050.123.456">
...
```

Il tipo `NOTATION` serve per identificare il formato di dati esterni che vogliamo collegare al documento XML, come ad esempio nel caso di immagini:

```
<!NOTATION GIF SYSTEM "gifview.exe"> <!NOTATION JPG SYSTEM
"jpgview.exe">
```

in questo modo si specifica l'associazione del formato immagini `GIF`, `JPG` con il programma corrispondente `gifview.exe` in grado di visualizzarli. In questo modo possiamo definire un attributo utilizzando la parola chiave `NOTATION`. Per esempio:

```
...
<!ATTLIST Immagine formato NOTATION (GIF|JPG) "GIF">
...
<Immagine tipo="JPG">
...
```

La dichiarazione di sopra specifica che l'elemento `Immagine` ha un attributo *`formato`* di tipo `NOTATION` e che i valori accettabili per questo attributo sono `GIF` e `JPG`. Se non viene specificato alcun tipo viene assunto `GIF`.

L'ultimo tipo di attributo è `Enumerated`, utile nel caso si voglia specificare una lista chiusa di valori per un attributo. Ad esempio:

```
<!ATTLIST Impiegato
        manager (vero | falso) #REQUIRED>

<!ATTLIST Finestra
        colore (bianco | nero | rosso) #REQUIRED>
```

Il gruppo dei valori permessi consiste in un lista di nomi separati dall'operatore `|`.

2.4 I linguaggi XML

XML è una tecnologia che serve a creare linguaggi di markup che descrivono i dati di qualsiasi tipo e in modo strutturato. XML è quindi un metalinguaggio. I linguaggi di markup creati utilizzando XML sono chiamati *applicazioni* o *vocabolari*. Alcuni famosi esempi di applicazioni XML sono:

- Mathematical Markup Language (MathML) definisce un linguaggio per la matematica;

- Chemical Markup Language (CML) definisce un linguaggio per la chimica;
- Channel Definition Format (CDF) utilizzato come formato aperto per scambiare informazioni sui canali;
- Open Software Description (OSD) utilizzato per descrivere il software;
- Sincronized Multimedia Integration Language (SMIL) utilizzato per descrivere elementi multimediali.

Alcuni linguaggi XML sono molto importanti in quanto forniscono strumenti per il trattamento e la rappresentazione di documenti XML stessi. Ad esempio, vi sono linguaggi per la rappresentazione di schemi XML, il più importante di questi è l'XML Schema del W3C (XSD), il cui scopo è quello di sostituire il DTD.

Uno dei principali vantaggi dell'XML Schema rispetto ai DTD di XML 1.0 è l'esistenza dei tipi di dati. Infatti, la possibilità di utilizzare, come valori di elementi o attributi, stringhe, date o dati numerici, consente agli autori di schemi di specificare e convalidare il contenuto dei dati XML in maniera interoperabile e indipendente dalla piattaforma. Ad esempio il contenuto di un elemento `<telefono>` con DTD deve essere specificato come `#PCDATA`, ossia qualsiasi testo: un documento contenente ad esempio `<telefono>ABC21</telefono>` verrebbe tranquillamente validato dal parser senza segnalare errori. Gli schemi promettono di risolvere questi problemi garantendo una validazione più potente introducendo ad esempio dei tipi di dato, permettendo di specificare quindi che il contenuto dell'elemento è un numero intero. Inoltre gli XML Schema sono essi stessi dei documenti XML, mentre i DTD sono un qualcosa di "estraneo", vengono definiti con un linguaggio a parte.

Il vantaggio di avere gli XML Schema scritti in XML è quello di poter utilizzare su di essi tutti gli strumenti standard per XML, quali parser, validatori, etc.

La trattazione di XML Schema sarebbe molto lunga, cerchiamo di fare un piccolo esempio per capire di cosa si tratta senza la pretesa di essere esaustivi. Riprendiamo il nostro esempio iniziale, aggiungendo un nuovo elemento chiamato `telefono`:

```
<persona sesso="maschio">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
    <telefono>3333334444</telefono>
```

```
</persona>
```

ecco come potrebbe essere un XML Schema che validi tale documento:

```
<?xml version="1.0" encoding="UTF-8"?> <xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="persona">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nome" type="xs:string"/>
                <xs:element name="cognome" type="xs:string"/>
                <xs:element name="telefono" type="xs:long"/>
            </xs:sequence>
            <xs:attribute name="sesso" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

La seconda riga `<xs:schema...>` specifica come `xs` lo spazio dei nomi di XML Schema. In pratica tutti gli elementi del linguaggio XML Schema saranno preceduti da prefisso "`xs:`". Il resto dell'XML Schema definisce uno schema per il documento di sopra. Nonostante la maggior complessità rispetto al DTD, XML Schema risulta più intuitivo. L'elemento `xs:element` definisce l'elemento `persona` attraverso il suo attributo `name`. All'interno di questo elemento compare un elemento di tipo `xs:complexType` che specifica che il contenuto dell'elemento persona è di tipo complesso, vale a dire composto di altri elementi e non solo di testo.

All'interno di `xs:complexType` viene specificata la sequenza degli sotto–elementi di cui è composto `persona` e i suoi attributi. In particolare con `xs:sequence` si specifica la lista dei sotto–elementi, che sono per l'appunto `nome`, `cognome` e `telefono`. Anche in questo caso l'attributo `name` ne specifica il nome, e inoltre attraverso l'attributo type viene specificato il tipo di dato all'interno dei sotto–elementi. Nel caso di `nome` e `cognome` è stato utilizzato il tipo di dato predefinito di XML Schema `xs:string` (cioè stringa), invece per il sotto–elemento `telefono` è stato utilizzato `xs:long`, cioè un intero lungo.

L'attributo dell'elemento persona viene poi definito con `xs:attribute` che definisce anche in questo caso il tipo di dato (`xs:string`), e che si tratta di un attributo obbligatorio attraverso l'attributo `use`.

Nell'esempio illustrato abbiamo visto gli elementi di base della creazione di XML Schema. Questo strumento prevede numerose altre caratteristiche

che lo rendono un metodo flessibile e potente per descrivere le regole di validità di un documento XML.

Ad esempio, è possibile dichiarare tipi di dato personalizzati e fare riferimento a tali dichiarazioni quando si definisce un elemento, in modo analogo a come avviene per la definizione di tipi nei linguaggi di programmazione. Questo consente di rendere più leggibile lo schema e di concentrare in un unico punto la definizione di un tipo utilizzato diverse volte.

2.5 Un esempio completo

In questa sezione presentiamo un esempio di un documento XML completo e il suo corrispondente DTD. Si tratta di un esempio di metadati del Dlib-Magazine, una rivista elettronica presente sul WEB focalizzata sulla ricerca e sviluppo nel campo delle biblioteche digitali. Qui di seguito è mostrato il documento XML contenente i metadati di un articolo dal titolo "Digital Libraries and the Problem of Purpose".

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-meta01.dtd">
<dlib-meta0.1>
    <title>Digital Libraries and the Problem of Purpose</title>
    <creator>David M. Levy</creator>
   <publisher>Corporation for National Research Initiatives</publisher>
    <date date-type = "publication">January 2000</date>
    <type resource-type = "work">article</type>
    <identifier uri-type = "DOI">
        10.1045/january2000-levy
    </identifier>
    <identifier uri-type = "URL">
        http://www.dlib.org/dlib/jan00/01.html
    </identifier>
    <language>English</language>
    <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
    </relation>
    <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Vediamo adesso il DTD corrispondente, che, come si nota dal `DOCTYPE` nel documento XML è un file esterno recuperabile all'indirizzo: "http://www.dlib.org/dlib/dlib-meta01.dtd".

Prima di tutto dobbiamo definire l'elemento radice `dlib-meta0.1`:

```
<!ELEMENT dlib-meta0.1 (title, creator+, publisher,
    date, type, identifier+, language*, relation, rights+)>
```

Esso è costituito da un elemento `title`, uno o più elementi `creator`, un elemento `publisher`, `date` e `type`, uno o più elementi `identifier`, zero o più elemento `language`, un elemento `relation` e in fine uno più elementi `rights`.

Tutti gli elementi, ad eccezione di `relation` non contengono sotto–elementi ma solo testo (`#PCDATA`). Ad esempio `title`, `creator` e `publisher` sono definiti come:

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT creator (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
```

Gli elementi `date` e `type` hanno degli attributi "fissi" non obbligatori:

```
<!ELEMENT date (#PCDATA)> <!ATTLIST date
          date-type CDATA #FIXED "publication">

<!ELEMENT type (#PCDATA)> <!ATTLIST type
          resource-type CDATA #FIXED "work">
```

L'elemento `identifier` possiede l'attributo `uri-type` obbligatorio che può assumere i valori `DOI` oppure `URL`.

```
<!ELEMENT identifier (#PCDATA)>
<!ATTLIST identifier
          uri-type (DOI | URL) #REQUIRED>
```

L'elemento `relation` è il più complesso, contiene un sotto–elemento `serial-name` (di tipo `#PCDATA`) e zero o più gruppi di sotto-elementi `issn`, `volume`, `issue` di tipo testuale. Inoltre `relation` possiede un attributo fisso di tipo testuale.

```
<!ELEMENT relation (serial-name, (issn, volume, issue)*)>
<!ATTLIST relation
```

```
          rel-type CDATA #FIXED "InSerial">
    <!ELEMENT serial-name (#PCDATA)>
    <!ELEMENT issn (#PCDATA)>
    <!ELEMENT volume (#PCDATA)>
    <!ELEMENT issue (#PCDATA)>
```

Infine il DTD definisce gli elementi `language` e `rights`:

```
<!ELEMENT language (#PCDATA)>
<!ELEMENT rights (#PCDATA)>
```

Qui di seguito viene mostrato il DTD di *D-Lib Magazine* completo con i commenti originali:

```
<!--  A draft DTD to mark up the metadata elements in D-Lib Magazine -->
<!--  William Y. Arms, Cathy Rey, March 8, 1999 Updated June 16, 1999 -->

<!ELEMENT dlib-meta0.1 (title, creator+, publisher,
    date, type, identifier+, language*, relation, rights+)>

<!--  These element names are from the Dublin Core set of 15 names.  -->
<!--  Attributes are used to clarify the usage by D-Lib Magazine.  -->

<!ELEMENT title (#PCDATA)>

<!--  Title as supplied with all punctuation -->

<!ELEMENT creator (#PCDATA)>

<!--  This element is repeated for each author or other creator -->
<!--  It contains the name of the author as provided, -->
<!--  without affiliation or contact information.  -->

<!ELEMENT publisher (#PCDATA)>

<!-- The publisher is "Corporation for National Research Initiatives" -->

<!ELEMENT date (#PCDATA)>
<!ATTLIST date
          date-type CDATA #FIXED "publication">
```

```
<!--  This is the issue date, e.g., "July 1995", or "July/August 1998" -->

<!ELEMENT type (#PCDATA)>
<!ATTLIST type
          resource-type CDATA #FIXED "work">

<!--  D-Lib Magazine assigns metadata to works -->
<!--  The default type is an "article" -->

<!ELEMENT identifier (#PCDATA)>
<!ATTLIST identifier
          uri-type (DOI | URL) #REQUIRED>

<!--  Every work should have a single DOI and one or more URLs. -->

<!ELEMENT relation (serial-name, (issn, volume, issue)*)>
<!ATTLIST relation
          rel-type CDATA #FIXED "InSerial">
    <!ELEMENT serial-name (#PCDATA)>
    <!ELEMENT issn (#PCDATA)>
    <!ELEMENT volume (#PCDATA)>
    <!ELEMENT issue (#PCDATA)>

<!--  The serial name is "D-Lib Magazine".  -->
<!--  The ISSN is "1082-9873".  -->
<!--  The volume corresponds to year of publication, with 1995 as "1".  -->
<!--  The issue is a count of the actual issues during in the volume.  -->
<!--  Thus the first issue "1" though published in July.  -->

<!ELEMENT language (#PCDATA)>

<!--  The name of the language in English as:
    "English", "French, "Japanese" -->

<!ELEMENT rights (#PCDATA)>

<!--  The copyright statement as given on the work.  -->
```